Automate your scans with these Perl scripts

At the Press of a Button

A Perl script creates PDFs from magazine articles by automating a process triggered by the simple press of a button. *By Michael Schilli*

Long-standing readers will be aware that this Perl column has been around for almost 10 years. What they probably don't know is that the collected paper issues of *Linux Pro* magazines in my apartment take up feet of valuable shelf space. I have considered renting a unit in a storage facility, but the rents in San Francisco definitely put me off that idea.

Before I recycle the mags, though, I would at least, for reasons of nostalgia, like to convert the articles into PDF format and store them in a database using the artscan script [1].

Fighting Fatigue

Scanning programs such as `xsane` [2] and `simple-scan` [3], a recent addition to Ubuntu (Figure 1), will handle individual scans without much ado. But, faced with the task of scanning several pages from a magazine and then composing the JPG images in a multipage PDF document, even the most ambitious scanner operators will soon feel the strain if they don't find a way to automate the process.

A newly created Perl script by the name of `artscan` guides you through the scanning process thanks to a terminal-based menubar, and it displays the step currently in progress in a list box in real time (Figure 2). As an additional benefit, you only need to press the green button on the scanner once you've set up the current page containing the article on the scanner bed (Figure 3).

To discard a series of individual images that you have already scanned, press the *N* (for "new") key in the terminal UI, which tells the script to ditch the scanner images that it is caching.

After scanning the last page in an article, you press the *F* (for "finish") key in the script's Curses front end. This calls the `convert` program from the ImageMagick suite to transform the cached pages from `.pnm` format to JPG images.

Downsizing with JPG

JPG compression can reduce the amount of disk space needed for a scanned page by up to 90%. Another call to `convert` then bundles the JPG collection into a multipage PDF document and saves it in the preset output directory. The footer line in the terminal shows the path to the resulting document (Figure 2).

The user can either press the *S* key in the script to start scanning an individual page or just hit the green button on the scanner.

Press Me!

While the operator is working with the scanner and trying to line up the original despite the crease in the middle of the magazine, it would be inconvenient to press a key in the terminal to tell the script to trigger the scan.

Scanners like my Epson feature a green button next to the scanbed that returns a signal to the controlling computer via the USB interface; the good thing is, the computer can interpret the signal in any way you like.

The `scanbuttond` [4] package for Ubuntu contains a daemon that monitors any scanners you have plugged in

MIKE SCHILLI

Mike Schilli works as a software engineer with Yahoo! in Sunnyvale, California. He can be contacted at *mschilli@perlmeister.com*. Mike's homepage can be found at *http://perlmeister.com*.

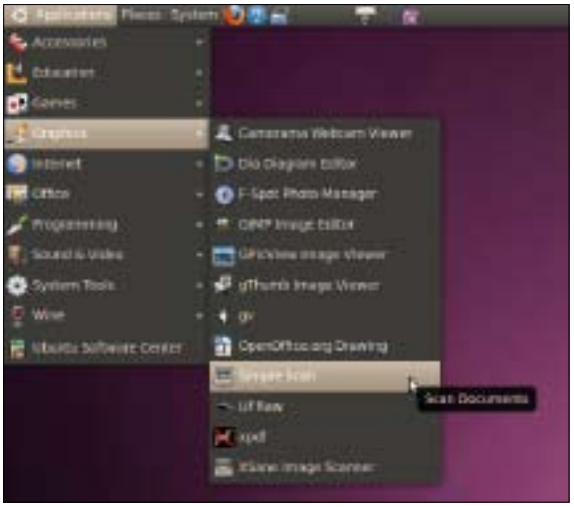

Figure 1: Newer Ubuntu distros include the Simple Scan program.

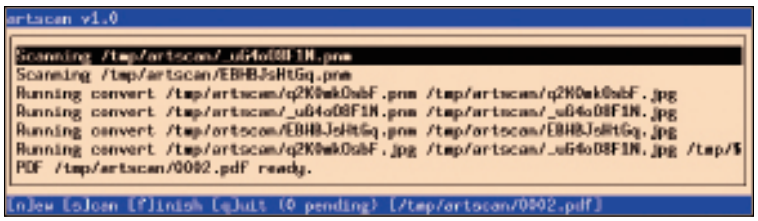

Figure 2: The program logs in the individual steps in a list box.

Mike: line 73?? -rls

and calls the preset `/etc/scanbuttond/buttonpressed.sh` script whenever the scanner button is pressed. The line

```
kill -USR1 `cat /tmp/pdfs/pid`
```

will send the Unix USR1 signal to the process whose PID is stored in the `/tmp/pdfs/pid`.

The `artscan` script (Listing 1) [5] does exactly this after launching with the use of the `blurt()` function from the Sysadm::Install module to write the PID available as `$$` in Perl to the `pid` file and adding a line break.

Dancing with POE

The `artscan` terminal front end dances to the beat of CPAN's POE framework, which you will be familiar with from previous Perl columns. The Curses::UI::POE module ties up the POE event loop with the Curses library, which draws the ASCII-based graphics elements on the terminal and reacts to keyboard input.

For reasons of space, the implementation doesn't follow the strict rules of the cooperative multitasking framework, which dictate that one task is never allowed to block another.

For example, long-running Unix commands like `convert` aren't allowed to block interrupts within the graphical user interface. But, because the user can't really do much besides wait for the scan or conversion to finish, the script doesn't worry about this and simply freezes the UI.

The `_start` handler defined in line 34 stores the POE session heap in the global `$HEAP` variable to allow the keyboard press handlers defined via `set_binding()` in line 71 to access the POE session data.

To make sure the program jumps to the `article_scan` handler when it receives the Unix signal triggered by the `scanbuttond` daemon, line 36 calls the POE kernel's `sig()` method and assigns the POE status `"article_scan"` to the signal. Line 41 defines the `article_scan` function (defined in line 163) as its jump address. Finally, when an asynchronously launched scan process completes, the kernel jumps to the third POE state, `"scan_finished"`.

The graphical interface builds on a window element defined in line 49 and consists of a bar at the top, `$TOP`, a list box `$LBOX` and a footer line, `$FOOT`. The script then uses `add()` to drop the widgets top-down into the main window. The footer line lands at the bottom of the window thanks to the `y -1` parameter pair; the width setting for the bar at the top, `-width -1`, causes the bar to use the full width of the terminal window.

Because of the binding in line 53, POE calls the `article_new()` function defined in line 83 when the *N* key is pressed. The function deletes any elements that exist in the global image array, `@IMAGES`, but only if the global `$BUSY` variable is not set. This action occurs in various parts of the program to prevent users triggering actions by pressing keys while a scan is in progress.

Mike: $FOOT->text()?? -rls

The script uses the `$FOOT->text()` method to report on current activities in the footer or the `lbox_add()` function to add an entry to the list box in the center. If the list box fills up the entire available screen real estate, it chops off any surplus elements at the top before adding more elements to the bottom to create the illusion of a scrolling file.

Tasks such as converting the raw .PNM-formatted data from the scanner to JPG are handled by the `task` function defined in line 153. It uses `tap` from the CPAN Sysadm::Install module to hand the arguments passed in to it to the shell.

The resulting PDF files are enumerated by the script (lines 147-149), starting at `0001.pdf`; the next value is discovered by

Mike: I added these line nos. to get the paragraph to wrap an extra line (the article was a little short). OK as set?? -rls

Figure 3: The scanner starts up at the push of a button and reads the cover image of a magazine from 1996.

searching the PDF directory for existing PDF files and incrementing the number of the last file.

Scanimage Workhorse

The CPAN Sane [6] module could handle the scan, but then the script would have to take care of all kinds of stuff, such as releasing the SANE [7] interface on terminating the program – failure to do this would cause any future scan attempts to hang.

Instead, the script takes the easy way out courtesy of the `scanimage` program included with the Sane package, which it launches by calling the `scan.sh` shell script.

As you can see from Listing 2, the resolution is set to 300 dpi, which should be fine for normal magazines. The `--mode` parameter uses the `Color` value to scan in color; my Epson scanner's default mode was monochrome. The shell script redirects the PNM-formatted raw data sent to stdout by `scanimage` into a file using the name passed to it by the Perl script.

Unless you pass the additional `x` and `y` parameters to it, the Epson scanner will only scan a small section of the available scanning area. The values of `1000` for `-x` and `-y` used by the shell script are reduced to the maximum available scanning area by the Sane back end, which just happens to be exactly the size of a computer magazine in the case of the Epson. You will need to experiment with these parameters for other scanner models or printed material.

Volatile Raw Data

To collect the raw data from the scanner, the script uses the CPAN File::Temp module and its exported `tempfile` function in line 168 of Listing 1 to create temporary files that automagically disappear shortly before `artscan` terminates, thanks to the `UNLINK` option.

The POE::Wheel::Run module calls the external scanner script `scan.sh`, which resides in the same directory. The

Figure 4: The finished article in PDF format after scanning.

Perl script launches a parallel process, calls the shell command with the temporary output file, and – thanks to the `CloseEvent` parameter – changes to the POE `scan_finished` POE state after completing the scan. All of this happens asynchronously, so that `new()` can return immediately in line 179.

To keep the wheel turning after exiting the `article_scan` function, line 187 stores the wheel data on the session heap. Line 192 then quickly writes "Scanning …" in the footer before the `article_scan` function terminates and control returns to the POE kernel, which then processes

Mike: 2 POEs OK here?? -rls

LISTING 1: artscan (continued on p 63)

```
001 #!/usr/local/bin/perl -w
002 ##############################
003 # artscan - Scan articles in
004 #           batches
005 # Mike Schilli, 2010
006 # (m@perlmeister.com)
007 ##############################
008 use strict;
009 use local::lib;
010 use POE;
011 use POE::Wheel::Run;
012 use Curses::UI::POE;
013 use Sysadm::Install qw(:all);
014 use File::Temp qw(tempfile);
015 use File::Basename;
016
017 my $PDF_DIR = "/tmp/artscan";
018 mkd $PDF_DIR
019   unless -d $PDF_DIR;
020
021 my $pidfile = "$PDF_DIR/pid";
022 blurt "$$\n", $pidfile;
023
024 my @LBOX_LINES = ();
025 my $BUSY       = 0;
026 my $LAST_PDF;
027 my @IMAGES = ();
028 my $HEAP;
029
030 my $CUI =
031   Curses::UI::POE->new(
032  -color_support => 1,
033  inline_states  => {
034   _start => sub {
035    $HEAP = $_[HEAP];
036    $_[KERNEL]->sig("USR1",
037     "article_scan");
038   },
039   scan_finished =>
040     \&scan_finished,
041   article_scan =>
042     \&article_scan,
043  }
044   );
045
046 my $WIN = $CUI->add("win_id",
047  "Window");
048
049 my $TOP = $WIN->add(
050  qw( top Label
051     -y 0 -width -1
052     -paddingspaces 1
053     -fg white -bg blue
054     ), -text => "artscan v1.0"
055 );
056
057 my $LBOX = $WIN->add(
058  qw( lb Listbox
059     -padtop 1
060     -padbottom 1 -border 1),
061 );
062
063 my $FOOT = $WIN->add(
064  qw( bottom Label
065     -y -1 -paddingspaces 1
066     -fg white -bg blue)
067 );
068
069 footer_update();
070
071 $CUI->set_binding(
072  sub { exit 0; }, "q");
073 $CUI->set_binding(
074  \&article_new, "n");
075 $CUI->set_binding(
076  \&article_scan, "s");
077 $CUI->set_binding(
078  \&article_finish, "f");
079
080 $CUI->mainloop;
081
082 ##############################
083 sub article_new {
084 ##############################
085  return if $BUSY;
086  @IMAGES = ();
087  footer_update();
088 }
089
090 ##############################
091 sub article_finish {
092 ##############################
093  return if $BUSY;
094  $BUSY = 1;
095
096  $FOOT->text(
097   "Converting ...");
098  $FOOT->draw();
099
100  my @jpg_files = ();
101
102  for my $image (@IMAGES) {
103   my $jpg_file = "$PDF_DIR/"
104     . basename($image);
105   $jpg_file =~
106     s/\.pnm$/.jpg/;
107   push @jpg_files, $jpg_file;
108   task("convert", $image,
109    $jpg_file);
110  }
111
112  my $pdf_file =
113    next_pdf_file();
114
115  $FOOT->text(
116   "Writing PDF ...");
117  $FOOT->draw();
118
119  task("convert", @jpg_files,
120   $pdf_file);
121  unlink @jpg_files;
122
123  $LAST_PDF = $pdf_file;
124  @IMAGES   = ();
125
126  lbox_add(
127   "PDF $LAST_PDF ready.");
128  footer_update();
129  $BUSY = 0;
130 }
131
132 ##############################
133 sub next_pdf_file {
134 ##############################
135  my $idx = 0;
136
137  my @pdf_files =
138    sort <$PDF_DIR/*.pdf>;
139
140  if (scalar @pdf_files > 0) {
141   ($idx) =
142     ($pdf_files[-1] =~
143      /(\d+)/);
144  }
145
146  return "$PDF_DIR/"
147    . sprintf("%04d",
```

subsequent events. Once the scanner has finally completed the scan, the wheel activates the `scan_finished` function in line 198 to remove the wheel data from the heap and to append the name of the temporary file with the raw data from the scanner to the global array, `@IMAGES` (lines 203-204).

Installation

Ubuntu Packages `imagemagick`, `libfile-temp-perl`, `libpoe-perl`, `libcurses-ui-perl`, and `libsysadm-install-perl` install the underpinnings that you need to get the script running. You need to make the tiny shell script, `scan.sh` (Listing 2), executable and store it in the same directory as the main `artscan` script.

If your distribution doesn't offer a Curses::UI::POE package, you will need to install this manually in a CPAN shell. If you use local::lib, the script will also need to include this, as shown in line 9 of `artscan`; if not, you can delete this line.

If you prefer to experiment with the scanner's Sane back end, I would recommend the CPAN Sane module, which is available as `libsane-perl` on Ubuntu.

Improvements

If you have a scanner with an automatic document feeder, you can make the scanning process even more efficient. Assuming you are willing to chop up the magazine with a strong pair of scissors or a guillotine, the scanner could automatically feed the pages one by one. A second scanning run would take care of the backs of the pages, and the script could reassemble the whole thing in the right order. Preserve those back issues for the next millennium! ■■■

LISTING 1: artscan (continued from p 62)

```
148   $idx + 1)
149    . ".pdf";
150 }
151
152 ##################
153 sub task {
154 ##################
155  my ($command, @ar
156
157  lbox_add("Running
158     . " @args");
159  tap($command, @ar
160 }
161
162 ############################
163 sub article_scan {
164 ############################
165  return if $BUSY;
166  $BUSY = 1;
167
168  my ($fh, $tempfile) =
169    tempfile(
170   DIR    => $PDF_DIR,
171   SUFFIX => ".pnm",
172   UNLINK => 1
173    );
174
175  lbox_add(
176   "Scanning $tempfile");
177
178  my $wheel =
179    POE::Wheel::Run->new(
180   Program     => "./scan.sh",
181   ProgramArgs => [$tempfile],
182   StderrEvent => 'ignore',
183   CloseEvent =>
184     "scan_finished",
185    );
186
187  $HEAP->{scanner} = {
188   wheel => $wheel,
189   file  => $tempfile
190  };
191
192  $FOOT->text(
193   "Scanning ... ");
194  $FOOT->draw();
195 }
196
197 ############################
198 sub scan_finished {
199 ############################
200  my ($heap) =
201    @_[ HEAP, KERNEL ];
202
203  push @IMAGES,
204    $heap->{scanner}->{file};
205  delete $heap->{scanner};
206  footer_update();
207  $BUSY = 0;
208 }
209
210 ############################
211 sub footer_update {
212 ############################
213  my $text =
214 "[n]ew [s]can [f]inish [q]"
215    . "uit ("
216    . scalar @IMAGES
217    . " pending)";
218
219  if (defined $LAST_PDF) {
220   $text .= " [$LAST_PDF]";
221  }
222  $FOOT->text($text);
223  $FOOT->draw();
224 }
225
226 ############################
227 sub lbox_add {
228 ############################
229  my ($line) = @_;
230
231  if (
232   scalar @LBOX_LINES
233   $LBOX->height() -
234  {
235   shift @LBOX_LINES
236  }
237  push @LBOX_LINES, $
238
239  $LBOX->{-values} =
240    [@LBOX_LINES];
241  $LBOX->{-labels} =
242    { map { $_ => $_ }
243     @LBOX_LINES };
244  $LBOX->draw();
245 }
```

Mike: I added these line nos. to get the paragraph to wrap (same reason as before). OK as set?? -rls

Judith: I messed with your Info box. -rls

LISTING 2: scan.sh

```
1 #!/bin/bash
2 scanimage -x 1000 -y 1000 \
3     --resolution=300 \
4     --mode Color >$1
```

INFO

[1] "Perl: Archiving PDFs" by Michael Schilli, *Linux Magazine*, June 2005, *http://www.linux-magazine.com/issue/55/Perl_Archiving_PDFs.pdf*

[2] XSane: *http://www.xsane.org/*

[3] Simple Scan: *https://launchpad.net/simple-scan*

[4] scanbuttond – A scanner button daemon for Linux: *http://scanbuttond.sourceforge.net*

[5] Listings for this article: *http://www.linuxpromagazine.com/Resources/Article-Code*

[6] Perl Sane module: *http://search.cpan.org/~ratcliffe/Sane-0.03/lib/Sane.pm*

[7] SANE: *http://www.sane-project.org/html*